Various Programming Hints

Peter Seiderer for http://www.ciselant.de

February 23, 2012

1 Qt qmake And Subversion svnversion

Incorporate a global Subversion[1] revision number as build/revision version in a Qt[2] project.

Subversion has the ability to substitute keywords[3] e.g. the keyword 'Revision' in a file will be expanded to the last known revision in which the file changed. If you want some GlobalRev as build/revision number in your project some extra work has to be done using the Subversion tool svnversion[4].

The following describes four different solutions for a Qt qmake based project checked into a Subversion repository. The first three solutions are based on a generated header 'revision.h' included from the main file 'example.cpp'. The fourth is based on a define directive at compile time.

1.1 Include file approaches

All three solutions are based on the same simple example files. A simple include file version.h (with a given define for the version string):

```
#ifndef version_h
#define version_h
#define VERSION "1.2.3"
#endif
```

A C++ file example.cpp with the main entry point printing the string 'version: <version>-build-<revision>' to stdout:

```
#include "version.h"
#include "revision.h"
#include <stdio.h>

```

[1] http://subversion.apache.org

[2] http://qt.nokia.com

[3] http://svnbook.red-bean.com/en/1.7/svn.advanced.props.special.keywords.html

[4] http://svnbook.red-bean.com/en/1.7/svn.ref.svnversion.re.html

```
int main(int argc, char* argv []) {
    printf("version: %s-build-%s\n", VERSION, REVISION);
    return 0;
}
```

1.1.1 Simple solution (update on every build)

The qmake project file example1.pro creates an up to date revision.h file with svnversion generated revision string on every build:

```
PRE_TARGETDEPS += $$OUT_PWD/revision.h

HEADERS = version.h
SOURCES = example.cpp
TARGET = example

QMAKE_EXTRA_TARGETS += revtarget
revtarget.target = $$OUT_PWD/revision.h
revtarget.commands = @echo \"updating file $$revtarget.target\"; \
  echo -e \"/* generated file (do not edit) */\\n\" \
    \"$$LITERAL_HASH ifndef revision_h\\n\" \
    \"$$LITERAL_HASH define revision_h 1\\n\" \
    \"$$LITERAL_HASH define REVISION \\\"`svnversion -n $$PWD`\\\"\\n\" \
    \"$$LITERAL_HASH endif\" > $$revtarget.target
revtarget.depends = FORCE
QMAKE_DISTCLEAN += $$revtarget.target
```

Example of a generated revision.h file:

```
/* generated file (do not edit) */
# ifndef revision_h
# define revision_h 1
# define REVISION "828"
# endif
```

Example output:

```
version: 1.2.3-build-828
```

for a subversion working directory up to revision r828, or

```
1.2.3-build-828M
```

with local modifications.

The downside of this solution is the every time rebuild of the executable because of the generated file ’revision.h’.

1.1.2 Better solution (update only in case a relevant file changed)

The main thing changed is line 15, the dependency is changed from '*FORCE*' to '*$$SOURCES $$HEADERS $$FORMS $$PWD/example2.pro*' (mind the '$$PWD' changes which are needed for an out of tree build). Now a new revision.h is generated each time one of the relevant source/header/forms/project file is changed, see example2.pro:

```
1  PRE_TARGETDEPS += $$OUT_PWD/revision.h
2
3  HEADERS = $$PWD/version.h
4  SOURCES = $$PWD/example.cpp
5  TARGET = example
6
7  QMAKE_EXTRA_TARGETS += revtarget
8  revtarget.target = $$OUT_PWD/revision.h
9  revtarget.commands = @echo \"updating file $$revtarget.target\"; \
10   echo -e \"/* generated file (do not edit) */\\n\" \
11     \"$$LITERAL_HASH ifndef revision_h\\n\" \
12     \"$$LITERAL_HASH define revision_h 1\\n\" \
13     \"$$LITERAL_HASH define REVISION \\\"`svnversion -n $$PWD`\\\"\\n\" \
14     \"$$LITERAL_HASH endif\" > $$revtarget.target
15 revtarget.depends = $$SOURCES $$HEADERS $$FORMS $$PWD/example2.pro
16 QMAKE_DISTCLEAN += $$revtarget.target
```

1.1.3 Perfect solution (update every time the svn status changes)

The qmake project file example3.pro:

```
PRE_TARGETDEPS += $$OUT_PWD/revision.h

HEADERS = version.h
SOURCES = example.cpp
TARGET = example

LITERAL_DOLLAR = $
QMAKE_EXTRA_TARGETS += revtarget
revtarget.target = $$OUT_PWD/revision.h
revtarget.commands = @SVNVERSION=\\\"`svnversion -n $$PWD`\\\"; \
  SVNFILEVERSION=`if [ -e $$revtarget.target ]; then \
      grep REVISION $$revtarget.target | \
      awk \'{ print $${LITERAL_DOLLAR}$${LITERAL_DOLLAR}4 }\' ; \
    else echo unknown; fi;`; \
  if [ $${LITERAL_DOLLAR}$${LITERAL_DOLLAR}SVNVERSION == \
    $${LITERAL_DOLLAR}$${LITERAL_DOLLAR}SVNFILEVERSION ]; \
```

```
    then \
      echo \"svn status not changed - keep file $$revtarget.target\"; \
    else \
      echo \"svn status changed - updating file $$revtarget.target\"; \
      echo -e \"/* generated file (do not edit) */\\n\" \
        \"$$LITERAL_HASH ifndef revision_h\\n\" \
        \"$$LITERAL_HASH define revision_h 1\\n\" \
        \"$$LITERAL_HASH define REVISION \\\"`svnversion -n $$PWD`\\\"\\n\" \
        \"$$LITERAL_HASH endif\" > $$revtarget.target; \
    fi
revtarget.depends = FORCE
QMAKE_DISTCLEAN += $$revtarget.target
```

This is the only solution which forces a recompile in case of a svn status update without file update (e.g. 'svn update' or 'svn commit').

1.2 Define directive approach

This one uses a slightly different file example.cpp (version.h is recycled unchanged) with changes to print a given defined 'REVISION' string (or 'unknown' otherwise):

```
#include "version.h"
#include <stdio.h>

#ifndef REVISION
#define REVISION  "unknown"
#endif

int main(int argc, char* argv []) {
    printf("version: %s-build-%s\n", VERSION, REVISION);
    return 0;
}
```

1.2.1 Simple define based solution

The define for 'REVISION' is set on compile time (see line 1) of the qmake project file example4.pro:

```
1   DEFINES += "REVISION=\\\"`svnversion -n $$PWD`\\\""
2
3   HEADERS = version.h
4   SOURCES = example.cpp
5   TARGET = example
6
```

This approach works only for very small projects (like the given example project) with only one source file. It will fail to update the revision string in case of a multi source file project if the main file does not depend on all other ones.